190031187 **NERELLA VENKATA RADHAKRISHNA**

OSD Practical-9

Pre-Lab:

1. Problems on page replacement and multilevel paging

Ans:

Problem:

Consider a system using multilevel paging scheme. The page size is 1 MB. The memory is byte addressable and virtual address is 64 bits long. The page table entry size is 4 bytes.

Find-

1. How many levels of page table will be required?
2. Give the divided physical address and virtual address.

Solution:

Given-

- Virtual Address = 64 bits
- Page size = 1 MB
- Page table entry size = 4 bytes

Number of Bits in Frame Number-

We have, Page table entry size = 4 bytes = 32 bits

Thus, Number of bits in frame number = 32 bits

Number of Frames in Main Memory-

We have, Number of bits in frame number = 32 bits

Thus,Number of frames in main memory = 2^{32} frames

Size of Main Memory-

Size of main memory = Total number of frames x Frame size

$= 2^{32} \times 1$ MB

$= 2^{52}$ B

Thus, Number of bits in physical address = 52 bits

Number of Bits in Page Offset-

We have,Page size = 1 MB = 2^{20} B

Thus, Number of bits in page offset = 20 bits

Alternatively,Number of bits in page offset

= Number of bits in physical address – Number of bits in frame number

= 52 bits – 32 bits

= 20 bits

Process Size-

Number of bits in virtual address = 64 bits

Thus,Process size = 2^{64} bytes

Number of Pages of Process-

Number of pages the process is divided = Process size / Page size

$= 2^{64}$ B / 1 MB

$= 2^{64}$ B / 2^{20} B

$= 2^{44}$ pages

Inner Page Table Size-

Inner page table keeps track of the frames storing the pages of process.

Inner page table size = Number of entries in inner page table x Page table entry size

= Number of pages the process is divided x Page table entry size

= 2^{44} x 4 bytes

= 2^{46} bytes

Now, we can observe-

- The size of inner page table is greater than the frame size (1 MB).
- Thus, inner page table can not be stored in a single frame.
- So, inner page table has to be divided into pages.

Number of Pages of Inner Page Table-

Number of pages the inner page table is divided = Inner page table size / Page size

$= 2^{46}$ B / 1 MB

$= 2^{46}$ B / 2^{20} B

$= 2^{26}$ pages

Now, these 2^{26} pages of inner page table are stored in different frames of the main memory.

Number of Page Table Entries in One Page of Inner Page Table-

Number of page table entries in one page of inner page table = Page size / Page table entry size

= 1 MB / 4 B

= 2^{20} B / 2^{2} B

= 2^{18} entries

Number of Bits Required to Search an Entry in One Page of Inner Page Table-

One page of inner page table contains 2^{18} entries.

Thus,

Number of bits required to search a particular entry in one page of inner page table = 18 bits

Outer Page Table-1 Size-

Outer page table-1 is required to keep track of the frames storing the pages of inner page table.

Outer page table-1 size

= Number of entries in outer page table-1 x Page table entry size

= Number of pages the inner page table is divided x Page table entry size

= 2^{26} x 4 bytes

= 2^{28} bytes

= 256 MB

Now, we can observe-

- The size of outer page table-1 is greater than the frame size (1 MB).
- Thus, outer page table-1 can not be stored in a single frame.
- So, outer page table-1 has to be divided into pages.

Number of Pages of Outer Page Table-1 -

Number of pages the outer page table-1 is divided = Outer page table-1 size / Page size

= 256 MB / 1 MB

= 256 pages

Now, these 256 pages of outer page table-1 are stored in different frames of the main memory.

Number of Page Table Entries in One Page of Outer Page Table-1

Number of page table entries in one page of outer page table-1 = Page size / Page table entry size

= 1 MB / 4 B

= 2^{20} B / 2^{2} B

= 2^{18} entries

Number of Bits Required to Search an Entry in One Page of Outer Page Table-1-

One page of outer page table-1 contains 2^{18} entries.

Thus,

Number of bits required to search a particular entry in one page of outer page table-1 = 18 bits

Outer Page Table-2 Size-

Outer page table-2 is required to keep track of the frames storing the pages of outer page table-1.

Outer page table-2 size = Number of entries in outer page table-2 x Page table entry size

= Number of pages the outer page table-1 is divided x Page table entry size

= 256 x 4 bytes

= 1 KB

Now, we can observe-

- The size of outer page table-2 is less than the frame size (16 KB).
- Thus, outer page table-2 can be stored in a single frame.
- In fact, outer page table-2 will not completely occupy one frame and some space will remain vacant.
- So, for given system, we will have three levels of page table.
- Page Table Base Register (PTBR) will store the base address of the outer page table-2.

Number of Bits Required to Search an Entry in Outer Page Table-2

Outer page table-2 contains 256 = 2^8 entries.

Thus,Number of bits required to search a particular entry in outer page table-2 = 8 bits

In-Lab

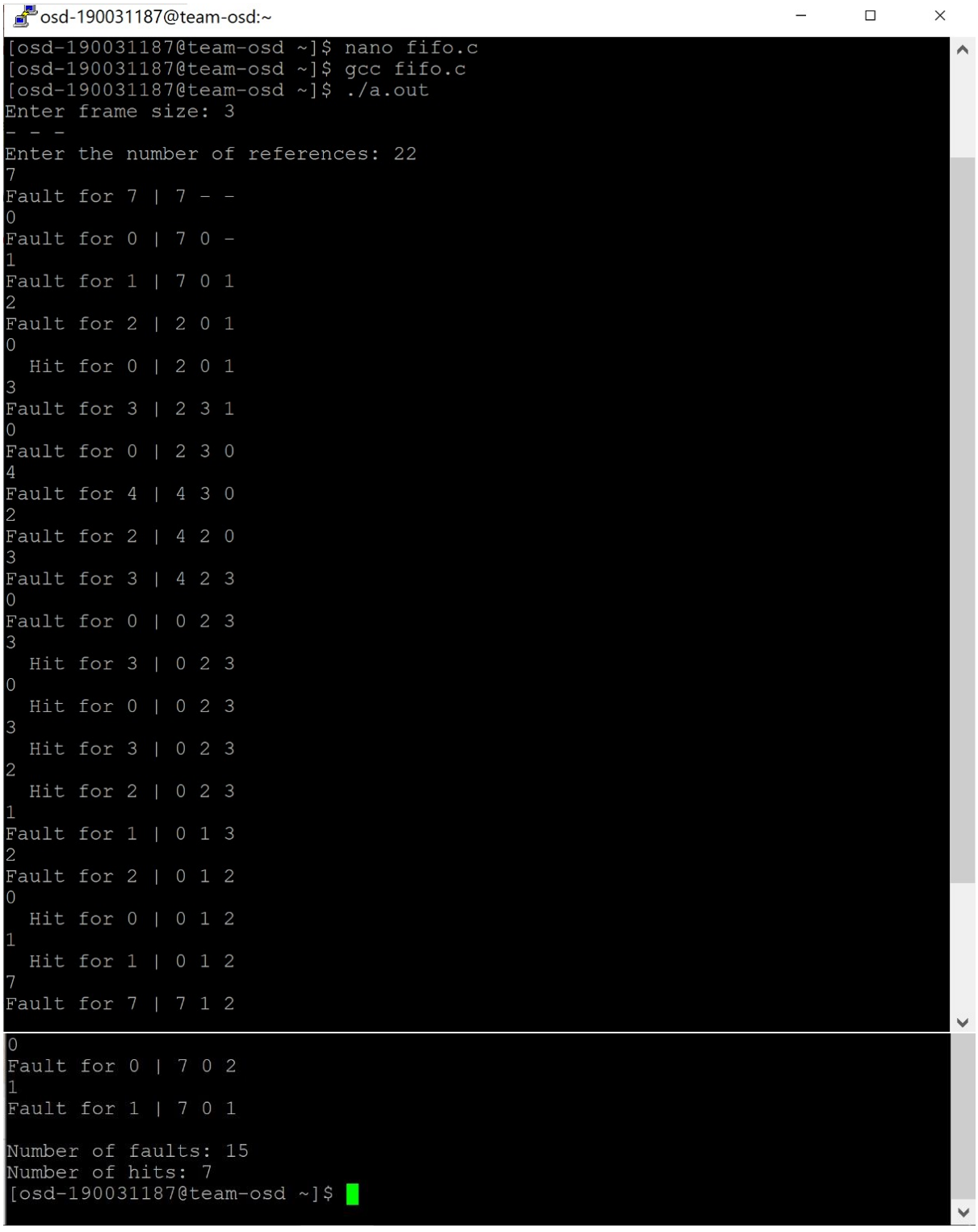
1. Write a program for the simulation of following paging algorithms FIFO LRU and MRU NFU.

Ans:

Fifo.c code

```
/*
                FIFO Page Replacement Algorithm
*/
#include "stdio.h"
#include "stdlib.h"
#include "stdbool.h"
 int
pointer;
int faults
,hits;
void print(int frame_size,int frame[]) {
                int i;
                    //printf("Printing the Frames: ");
                 for(i=0;i<frame_size;i++) {
                                if(frame[i]==-1)
                                  printf("- ");
                                else
                                  printf("%d ",frame[i]);
                 }
                printf("\n");
}
void add_reference(int frame_size,int frame[], int reference) {
                 int i;
                 bool alloted = false;
                 for(i=0;i<frame_size;i++) {
                                if(frame[i]==reference) {
                                                alloted = true;
                                                 printf("  Hit for %d | ", reference);
                                        hits++;
                                          break;
                                 }
                                 else if(frame[i]==-1) {
                                                alloted = true;
                                                 frame[i] = reference;
                                                 printf("Fault for %d | ", reference);
```

```
                                faults++;
                            break;
                        }
                }
                if(alloted == false) {
                        faults++;
                        printf("Fault for %d | ", reference);
                        frame[pointer] = reference;
                        pointer = (pointer+1)%frame_size;
                }
                print(frame_size, frame);
}
int main() {
 int frame_size,i,number_of_references;
printf("Enter frame size: ");
scanf("%d",&frame_size);
int frame[frame_size];
 for(i=0;i<frame_size;i++) {
          frame[i] = -1;
 }
print(frame_size,frame);
printf("Enter the number of references: ");
scanf("%d",&number_of_references);
 int reference[number_of_references];
for(i=0;i<number_of_references;i++) {
           scanf("%d",&reference[i]);
          add_reference(frame_size,frame,reference[i]);
  }
printf("\nNumber of faults: %d \nNumber of hits: %d\n",faults,hits );
}
```

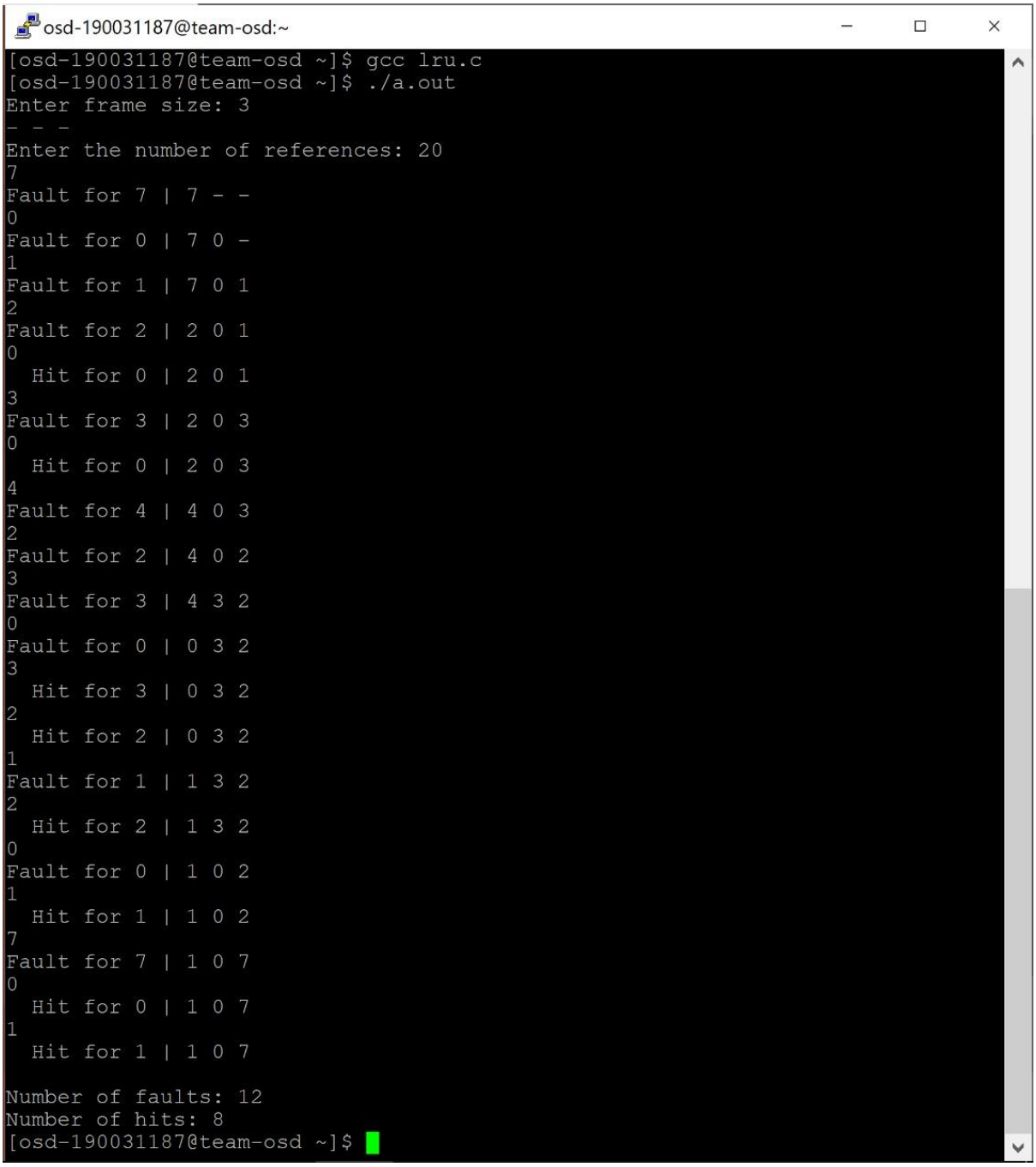


lru.c code:

```
/* Least Recently Used Page Replacement Algorithm */
#include "stdio.h"
#include "stdlib.h"
#include "stdbool.h"
int pointer;
```

```
int faults, hits;
void print(int frame_size,int frame[]) {
        int i;
        //printf("Printing the Frames: ");
        for(i=0;i<frame_size;i++) {
            if(frame[i]==-1)
                printf("- ");
            else
                printf("%d ",frame[i]);
        }
        printf("\n");
}
int predict(int reference_length, int references[], int page_no ,int frame_size,int frame[], int
start)
{
         int pos = -1, farthest = start, i;
         for(i=0;i<frame_size;i++)  {
                  int j;
                 for(j=start-1;j>=0;j--) {
                          if(frame[i]==references[j]) {
                                  if(j<farthest) {
                                          farthest=j;
                                                  pos=i;
                                  }
                                  break;
                           }
                  }
                 if(j==page_no)
                          return i;
         }
         if(pos == -1)  return 0;
          else   return pos;
}
void add_reference(int frame_size,int frame[], int reference, int
current_position,int reference_length, int references[])
{
         int i;
         bool allocated=false;
         for(i=0;i<frame_size;i++) {
             if(frame[i]==reference) {
                          printf("  Hit for %d | ", reference);
```

```
                    hits++;
                allocated = true;
                    break;
                }
                else if(frame[i]==-1) {
                        frame[i] = reference;
                        printf("Fault for %d | ", reference);
                        faults++;
                        allocated = true;
                        break;
                }
            }
            if(allocated==false) {
 int j =predict(reference_length,references,current_position,frame_size,frame,current
_position+1);
                frame[j] = reference;
                printf("Fault for %d | ", reference);
                faults++;
            }
            print(frame_size, frame);
}
int main() {
 int frame_size,i,number_of_references;
printf("Enter frame size: ");
scanf("%d",&frame_size);
int frame[frame_size];
 for(i=0;i<frame_size;i++)
     frame[i] = -1;
print(frame_size,frame);
printf("Enter the number of references: ");
scanf("%d",&number_of_references);
 int reference[number_of_references];
for(i=0;i<number_of_references;i++) {
            scanf("%d",&reference[i]);
            add_reference(frame_size,frame,reference[i],i,number_of_references,ref
erence);
 }
 printf("\nNumber of faults: %d \nNumber of hits: %d\n",faults,hits );
}
```


osd-190031187@team-osd:~
[osd-190031187@team-osd ~]$ gcc lru.c
[osd-190031187@team-osd ~]$ ./a.out
Enter frame size: 3
- - -
Enter the number of references: 20
7
Fault for 7 | 7 - -
0
Fault for 0 | 7 0 -
1
Fault for 1 | 7 0 1
2
Fault for 2 | 2 0 1
0
Hit for 0 | 2 0 1
3
Fault for 3 | 2 0 3
0
Hit for 0 | 2 0 3
4
Fault for 4 | 4 0 3
2
Fault for 2 | 4 0 2
3
Fault for 3 | 4 3 2
0
Fault for 0 | 0 3 2
3
Hit for 3 | 0 3 2
2
Hit for 2 | 0 3 2
1
Fault for 1 | 1 3 2
2
Hit for 2 | 1 3 2
0
Fault for 0 | 1 0 2
1
Hit for 1 | 1 0 2
7
Fault for 7 | 1 0 7
0
Hit for 0 | 1 0 7
1
Hit for 1 | 1 0 7
Number of faults: 12
Number of hits: 8
[osd-190031187@team-osd ~]$

Post-Lab:

1.Program to demonstrate Multi-level Page Table Control Flow.

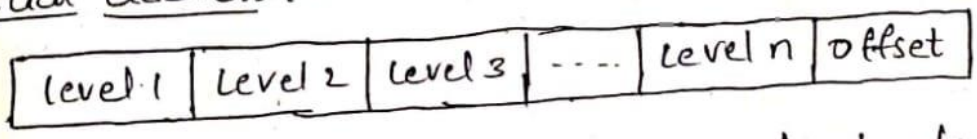
OSD practical-9

190031187
Radhakrishna

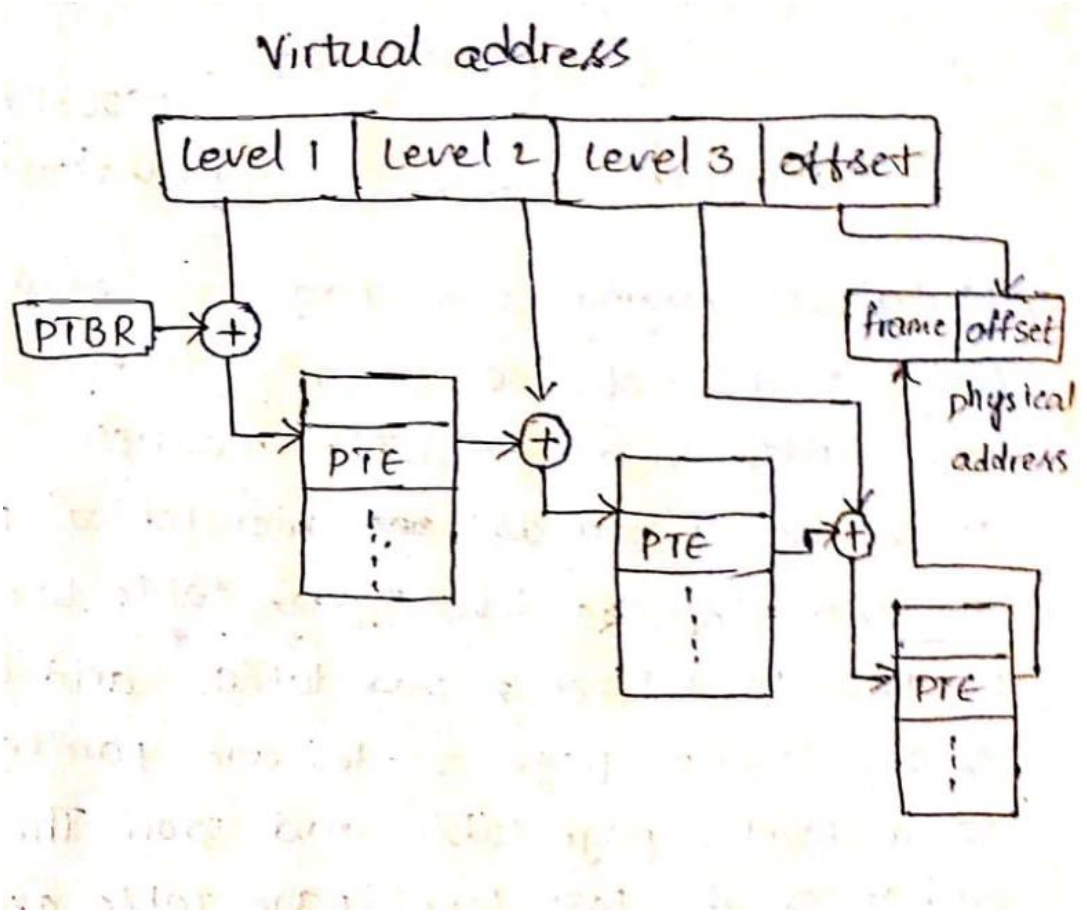
Post-lab

1, Multilevel paging is a paging schema which consist of two or more levels of page tables in a hierarchial manner
It is also known as hierarchical paging.
The entries of the level 1 page table are pointers to a level 2 page tables and entries of the level 2 page tables are pointers to a level 3 page table and soon. The entries of the last level page table are stores actual frame information.

virtual address :-



In multilevel paging whatever may be levels of paging all the page tables will be stored in main memory. so, it requires more than one memory access to get the physical address of page frame. one access for each level needed. Each page table entry except the last level page table entry contains base address of the next level page table.


Virtual address
level 1
level 2
level 3
offset
PTBR
PTE
PTE
PTE
frame
offset
physical address